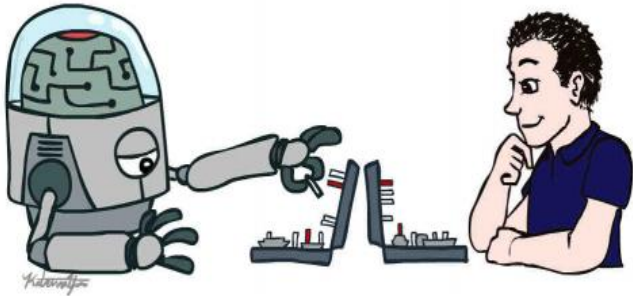
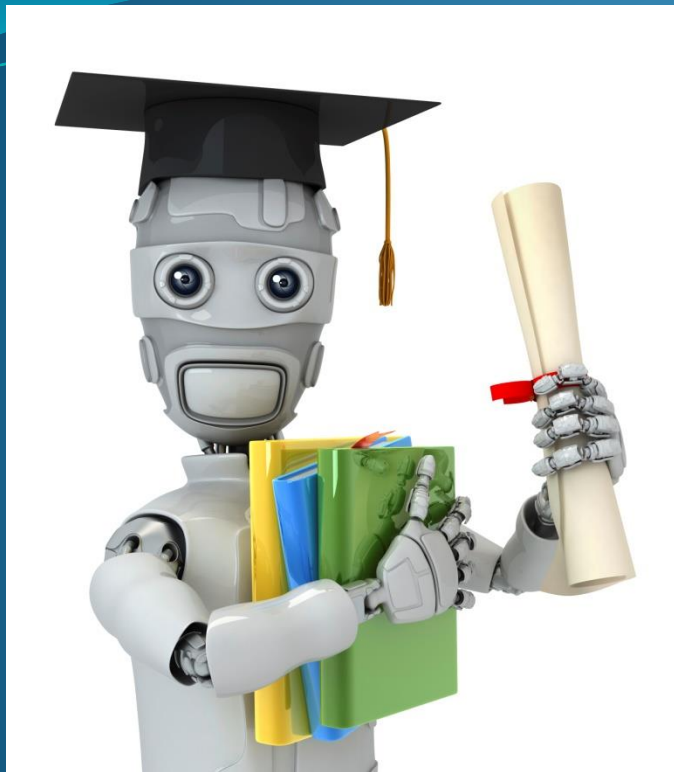


Artificial Intelligence

Part I: Machine learning





Machine Learning

Logistic Regression

Advanced Optimization Algorithms

Advanced Optimization Algorithms

- These algorithms are able to get logistic regression to run much more quickly than it's possible with gradient descent.
- They also scale much better to very large machine learning problems, such as if we had a very large number of features.

Advanced Optimization Algorithms

Cost function $J(\theta)$. Want $\min_{\theta} J(\theta)$.

Given θ , we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \dots, n$)

Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Optimization algorithm

Given θ , we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \dots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

Example:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
function [jVal, gradient]
    = costFunction(theta)
    jVal = (theta(1)-5)^2 + ...
           (theta(2)-5)^2;
    gradient = zeros(2,1);
    gradient(1) = 2*(theta(1)-5);
    gradient(2) = 2*(theta(2)-5);
```

```
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
    = fminunc(@costFunction, initialTheta, options);
```

fminunc attempts to find a minimum of a scalar function of several variables, starting at an initial estimate.

```
[optTheta, functionVal, exitFlag] ...  
    = fminunc(@costFunction, initialTheta, options);
```

The function arguments:

1. The objective function to be minimized (the cost function)
2. The initial values of the function variables (parameters θ)
3. The optimization options specified in options

The function return:

1. The optimal values of the function variables (parameters θ) at which the local minimum of the function.
2. The value of the objective function at the optimal values of the parameters.
3. A value exitFlag that describes the exit condition.

$$\text{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

```
function [jVal, gradient] = costFunction(theta)
```

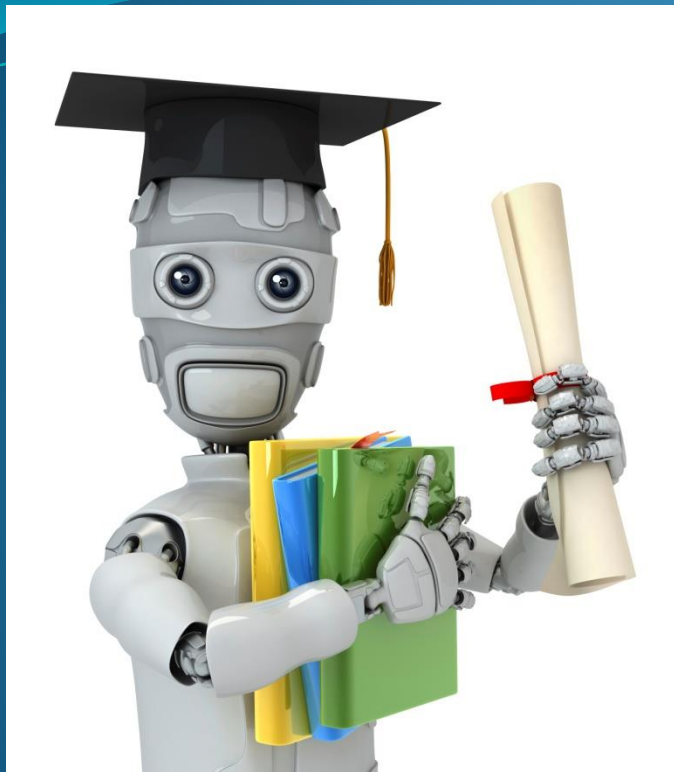
```
    jVal = [code to compute  $J(\theta)$ ];
```

```
    gradient(1) = [code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ];
```

```
    gradient(2) = [code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ];
```

```
    :
```

```
    gradient(n+1) = [code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$  ];
```

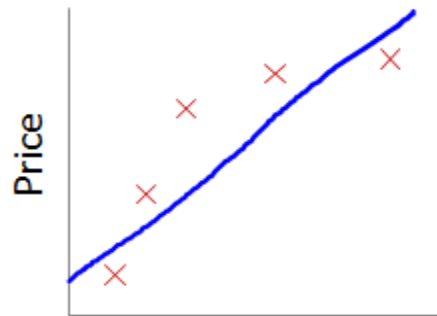



Machine Learning

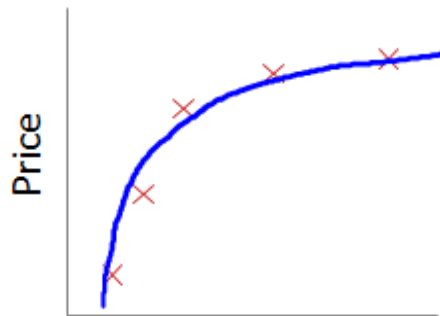
Regularization

The problem of overfitting

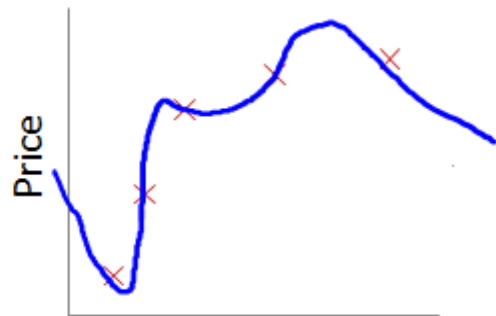
Example: Linear regression (housing prices)



Size
 $\theta_0 + \theta_1 x$
Under fitting



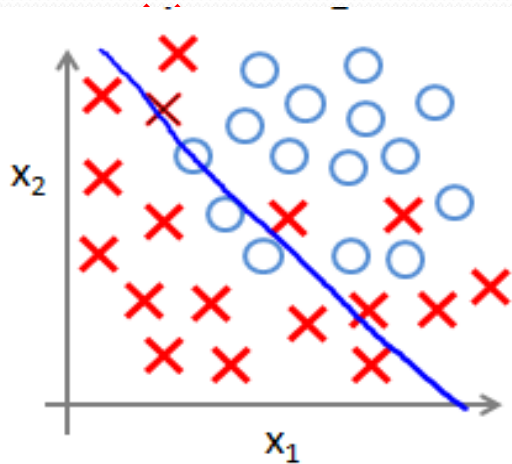
Size
 $\theta_0 + \theta_1 x + \theta_2 x^2$
Just Right



Size
 $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
Over fitting

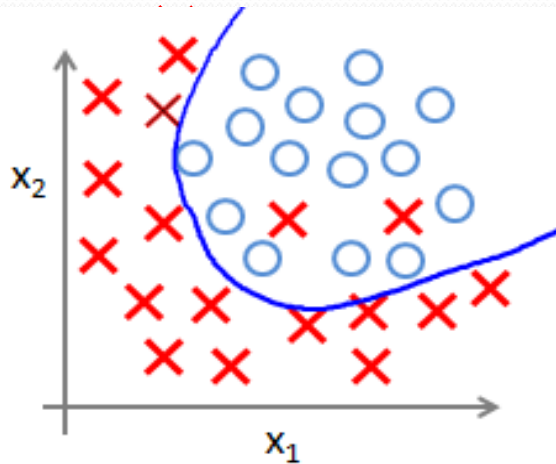
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Example: Logistic regression

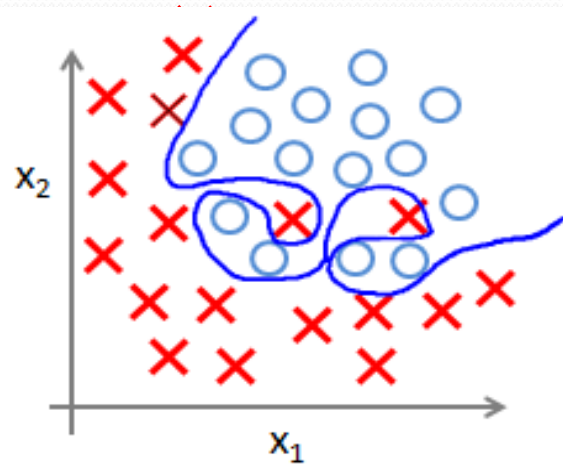


$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g = sigmoid function)



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

Addressing overfitting:

x_1 = size of house

x_2 = no. of bedrooms

x_3 = no. of floors

x_4 = age of house

x_5 = average income in neighborhood

x_6 = kitchen size

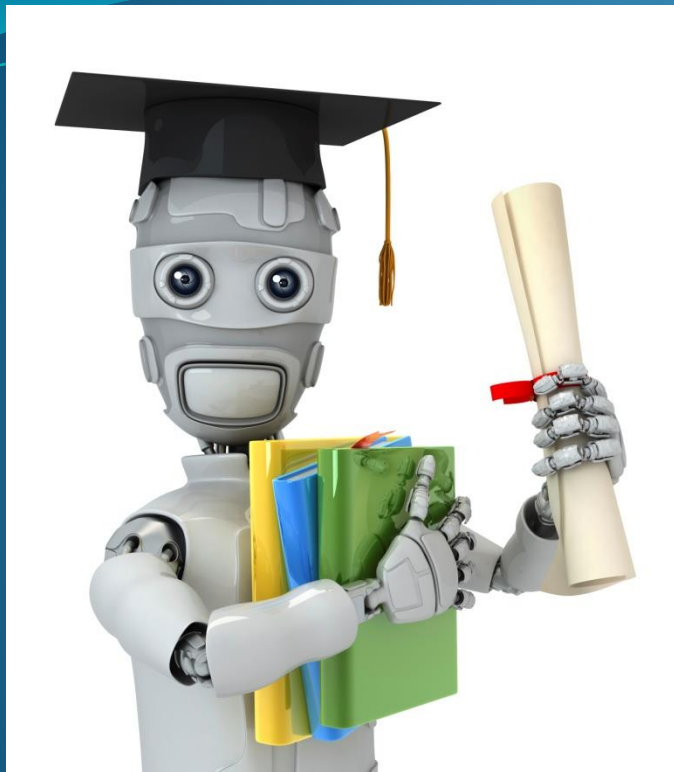
\vdots

x_{100}

Addressing overfitting:

Options:

1. Reduce number of features.
 - Manually select which features to keep.
 - Model selection algorithm.
 - Has disadvantage.
2. Regularization.
 - Keep all the features, but reduce magnitude/values of parameters θ_j .
 - Works well when we have a lot of features, each of which contributes a bit in predicting y .

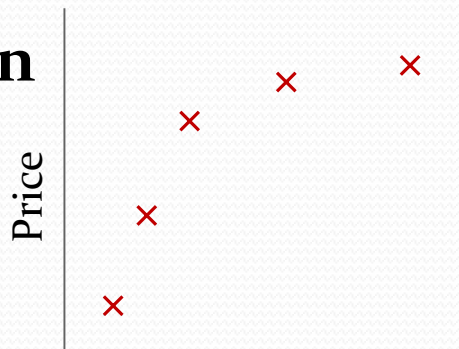


Machine Learning

Regularization

Cost function

Intuition



Size of house

$$\theta_0 + \theta_1 x + \theta_2 x^2$$



Size of house

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Suppose we penalize and make θ_3 , θ_4 really small.

Regularization.

Small values for parameters $\theta_0, \theta_1, \dots, \theta_n$

- “Simpler” hypothesis
- Less prone to overfitting
- Features: x_1, x_2, \dots, x_{100}
- Parameters: $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

Regularization.

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

$$\min_{\theta} J(\theta)$$

The extra regularization term at the end is used to shrink every single parameter θ (used to penalize the parameters θ_j).

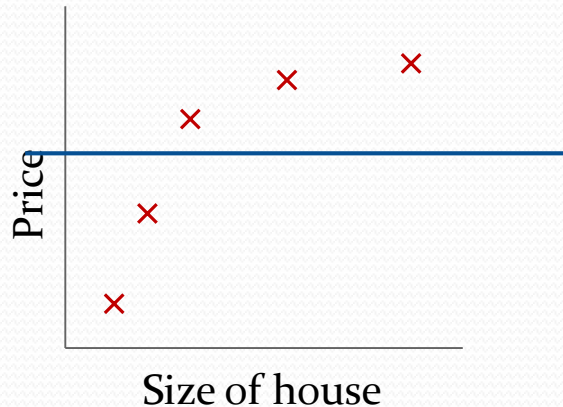
λ is the regularization parameter that controls a trade off between two different goals.

- The objective goal of fitting the training data well.
- And the goal of keeping the parameters small and therefore keeping the hypothesis relatively simple to avoid overfitting.

In regularized linear regression, we choose θ to minimize

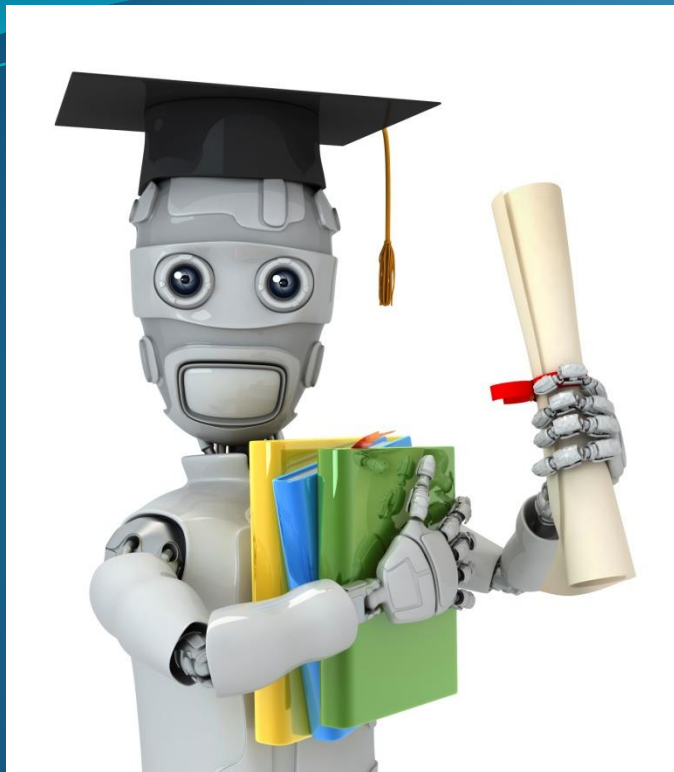
$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

What if λ is set to an extremely large value (means penalizing θ_j heavily)?....perhaps for too large, assume for our problem $\lambda = 10^{10}$



This means $\theta_1 \approx 0, \theta_2 \approx 0, \dots$,
so $h(\theta) = \theta_0$
Underfitting problem

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$



Machine Learning

Regularization

Regularized linear regression

Regularized linear regression

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

$$\min_{\theta} J(\theta)$$

Gradient descent

Repeat {

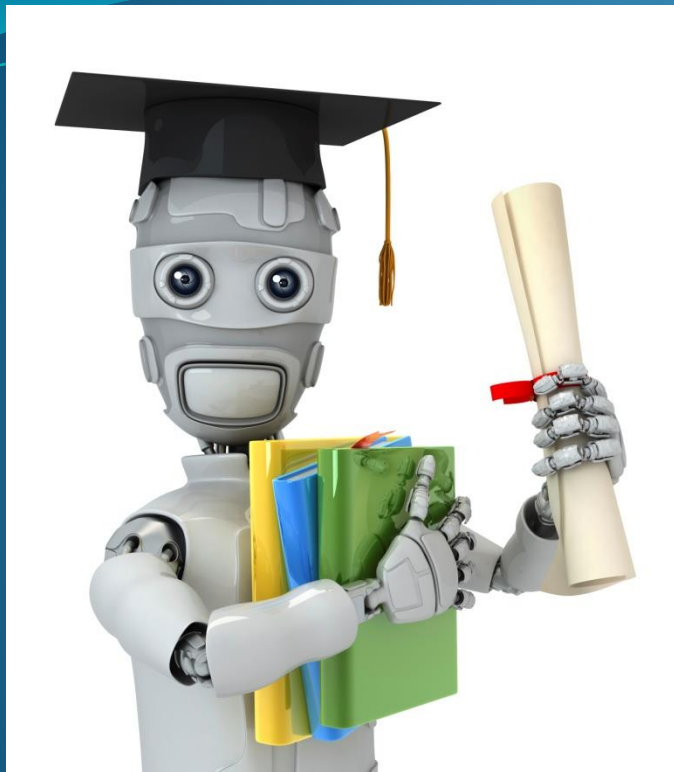
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

$(j = \text{red X}, 1, 2, 3, \dots, n)$

}

$$\theta_j := \theta_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

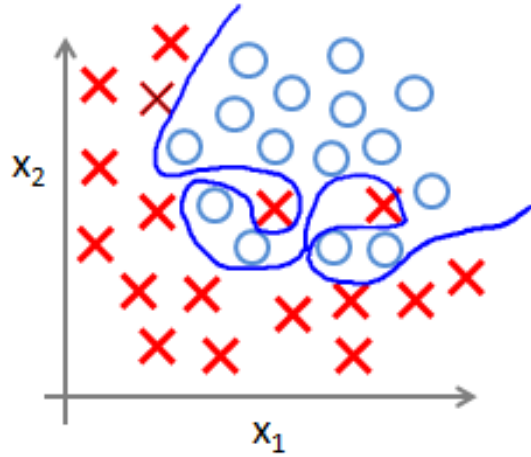


Machine Learning

Regularization

Regularized logistic regression

Regularized logistic regression.



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

$$J(\theta) = - \left[\frac{1}{m} \sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Gradient descent

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

$(j = \text{red X}, 1, 2, 3, \dots, n)$

}

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
```

```
jVal = [code to compute  $J(\theta)$ ] ;
```

$$J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

```
gradient(1) = [code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ] ;
```

$$\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

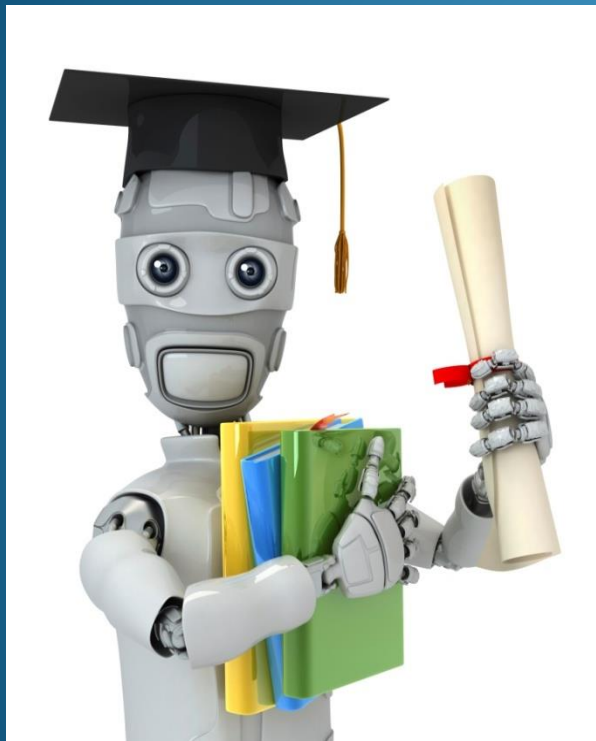
```
gradient(2) = [code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ] ;
```

$$\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)} - \frac{\lambda}{m} \theta_1$$

```
gradient(3) = [code to compute  $\frac{\partial}{\partial \theta_2} J(\theta)$ ] ;
```

$$\vdots \quad \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)} - \frac{\lambda}{m} \theta_2$$

```
gradient(n+1) = [code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$ ] ;
```



Thanks